Freescale Semiconductor
Application Note

AN2701
Rev. 0, 10/2004

PWM Generation with the HC08 Timer

by: Alban Rampon
8/16-bit Division
East Kilbride

Introduction

All HC08 microcontrollers (MCUs) include at least one timer module (TIM). This module is very useful for generating or capturing time-dependent signals.

The data sheet describes what the timer module is capable of and how to use it.

This application note gives further explanation of how the TIM behaves in the following situations.

- Unbuffered pulse width modulation (PWM) signal generation
- Rolling PWM generation
- Timer activity during Break and software interrupts

PWM signals are used in many applications from dimmers (duty cycle variation gives more/less light from LEDs) to IR transceivers (modulation/demodulation).

An example illustrating rolling generation is included, with sample code, in the appendix at the end of this document.

freescale™
semiconductor

Principle of PWM Generation Using Timers

The generation of a PWM signal using the dedicated HC08 PWM module is based on hardware comparisons between register values and a free-running hardware counter. The HC08 TIM offers similar hardware comparison in the form of output compare circuitry. The contents of channel registers are continuously compared to the master free-running timer. When a match occurs, a hardware output event can be configured to take place, and/or an interrupt can then call a service routine. The timer module has from two to six compare registers and can thus be configured to toggle the port pin on each of two to six comparison values, with no core overhead.

Each PWM signal requires a dedicated timer channel with output compare capability. Any channel with this function along with the associated interrupt vector may be used to generate a PWM signal, using the methods described.

Functional Description

Features of the TIM can include

- Two to six input capture/output compare channels
  - Rising-edge, falling-edge or any-edge input capture trigger
  - Set, clear or toggle output compare action
- Buffered and unbuffered pulse width modulation (PWM) signal generation
- Programmable TIM clock input
  - Seven frequency internal bus clock prescaler selection
  - External TIM clock input (4 MHz maximum frequency)
- Free-running or modulo up-count operation
- Toggle any channel pin on overflow
- TIM counter Stop and Reset bits

Block Diagram

Figure 1 shows the structure of the TIM, taken from an MC68HC908GZ60 (two channels represented out of the six available). The central component of the TIM is the 16-bit counter, which can operate as a free-running counter or a modulo up-counter. The TIM counter provides the timing reference for the input capture and output compare functions. The TIM counter modulo registers, TMODH:TMODL, control the modulo value of the timer counter. Software can read the TIM counter value at any time without affecting the counting sequence.

In the case of the HC908GZ60, the six timer channels are programmable independently as input capture or output compare channels. In other devices, the number of channels may vary from two to six; however, the rest of the module remains identical.

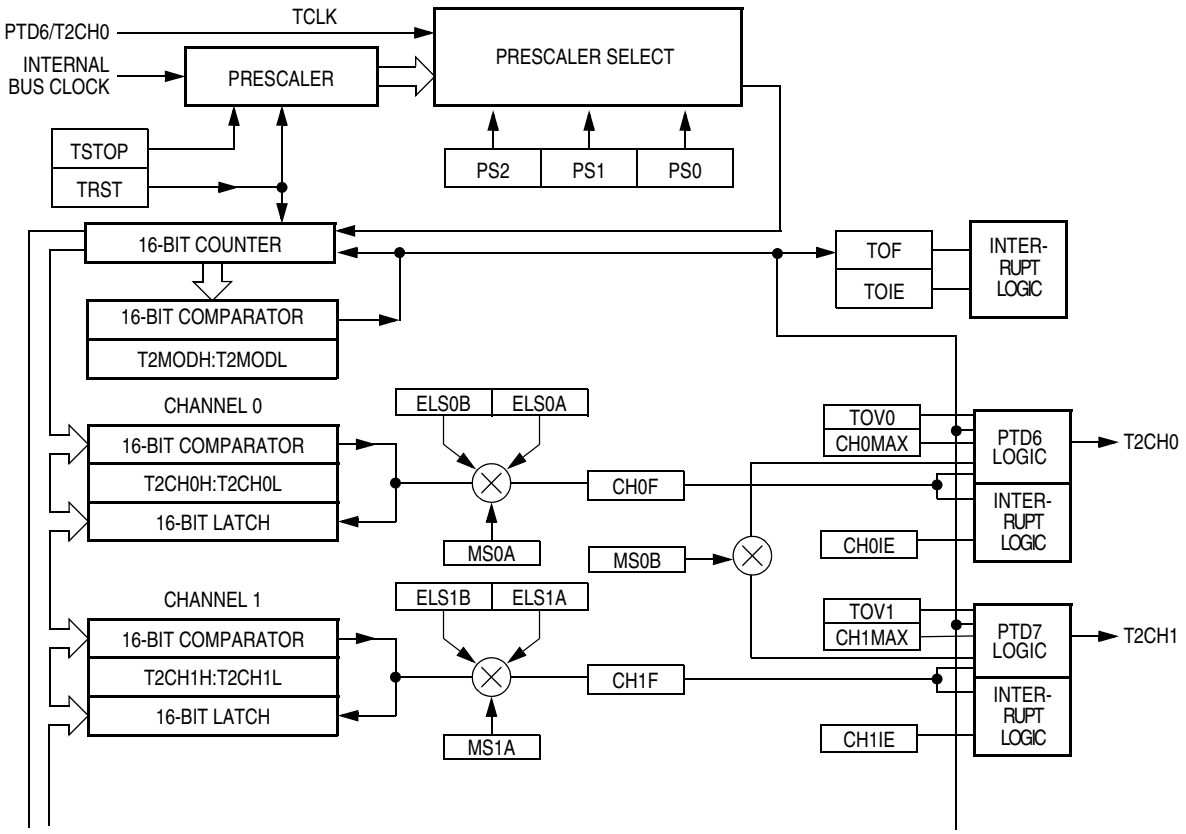


Figure 1. HC08 Generic Timer Module Block Diagram

Timer Counter Prescaler

The timer clock source can be one of the seven prescaler outputs or the timer clock pin (PTD6/T2CH0 on HC908GZ60). The prescaler generates seven clock rates from the internal bus clock. The prescaler select bits, PS[2:0], in the timer status and control register select the timer clock source (see Table 1).

Table 1. Timer Prescaler Values(1)

PS[2:0]	TIM2 Clock Source
000	Internal Bus Clock ÷ 1
001	Internal Bus Clock ÷ 2
010	Internal Bus Clock ÷ 4
011	Internal Bus Clock ÷ 8
100	Internal Bus Clock ÷ 16
101	Internal Bus Clock ÷ 32
110	Internal Bus Clock ÷ 64
111	T2CH0

NOTES:
1. Not all timers allow all possible values; refer to the device data sheet.

PWM Generation

Initialization Procedure

To ensure correct operation when generating unbuffered or buffered PWM signals, follow the procedure as stated in the data sheet:

1. In the TIMx status and control register (TxSC):
   a. Stop the TIMx counter by setting the TIMx stop bit, TSTOP.
   b. Reset the TIMx counter and prescaler by setting the TIMx reset bit, TRST.
2. In the TIMx counter modulo registers (TxMODH:TxMODL) write the value for the required PWM period.
3. In the TIMx channel y registers (TxCHyH:TxCHyL) write the value for the required pulse width.
4. In TIMx channel y status and control register (TxSCy):
   c. Write 0:1 (for unbuffered output compare or PWM signals) or 1:0 (for buffered output compare or PWM signals) to the mode select bits, MSxB:MSxA.
   d. Write 1 to the toggle-on-overflow bit, TOVx.
   e. Write 1:0 (to clear output on compare) or 1:1 (to set output on compare) to the edge/level select bits, ELSxB:ELSxA. The output action on compare must force the output to the complement of the pulse width level.

NOTE

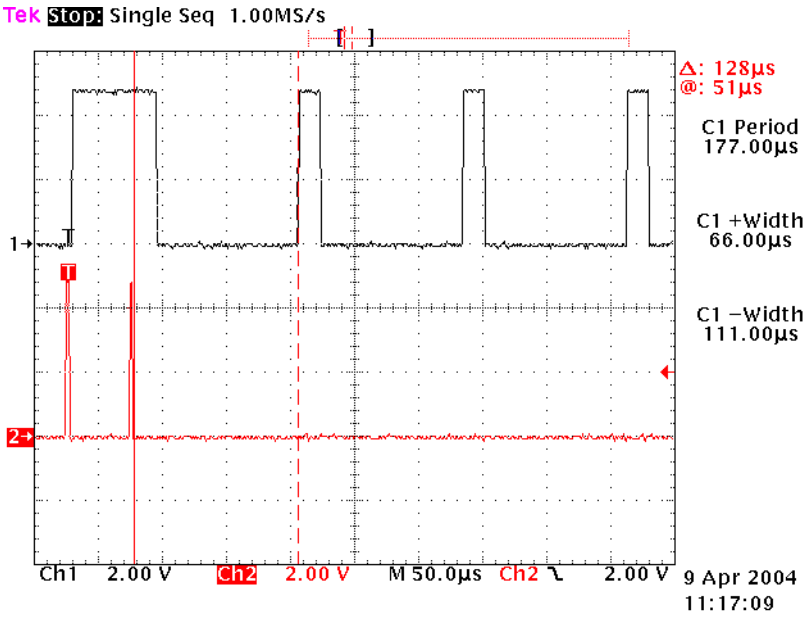
IMPORTANT: Do not use direct assignment instructions, such as MOV #0x1A, 0x25, or STA 0x25, for channel status and control registers.

Assignment must be done sequentially (using BSET 4,+ 0x25, for example) following the steps of the PWM Initialization procedure described above.

Table 2. Mode, Edge, and Level Selection

MSxB	MSxA	ELSxB	ELSxA	Mode	Configuration
x	0	0	0	Output Preset	Pin under port control; initial output level high
x	1	0	0		Pin under port control; initial output level low
0	0	0	1	Input Capture	Capture on rising edge only
0	0	1	0		Capture on falling edge only
0	0	1	1		Capture on rising or falling edge
0	1	0	0	Output Compare or PWM	Software compare only
0	1	0	1		Toggle output on compare
0	1	1	0		Clear output on compare
0	1	1	1		Set output on compare
1	x	0	1	Buffered Output Compare or Buffered PWM	Toggle output on compare
1	x	1	0		Clear output on compare
1	x	1	1		Set output on compare

If this procedure is not followed, the PWM signal generated will not be as expected.



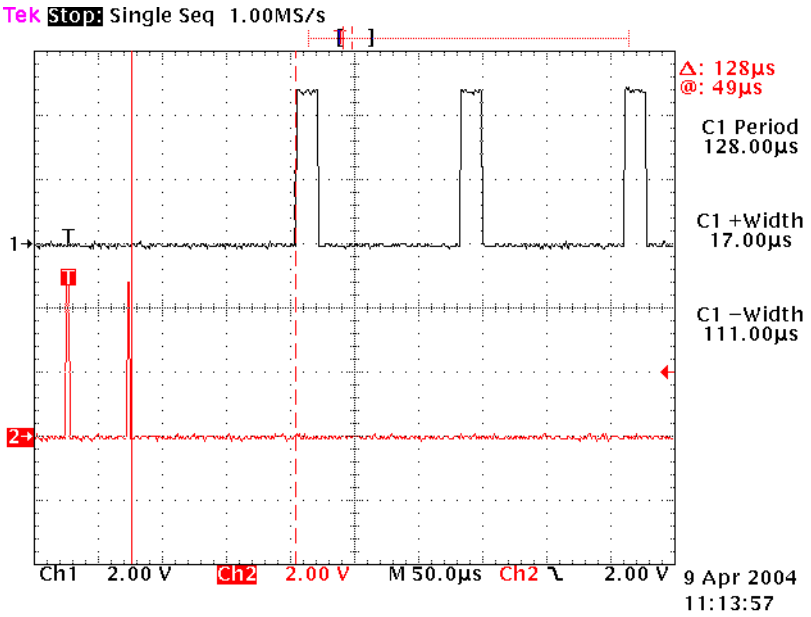
Channel 1: PWM signal generated
Channel 2: Events

Figure 2. Timer Erroneous Initialization

Figure 2 shows what the resulting PWM signal may look like. In this case, the full register has been written instead of setting the bits in proper order. The first pulse on Channel 2 is just after the direct assignment to the timer status and control register is complete. The second pulse follows the instruction to start the timer.

Between these two events, the software initializes other modules. The first pulse of the PWM appears to be quite long and earlier than expected. Its length also depends on the code executed between the initialization and the free-running counter startup.

Figure 3 shows the PWM signal being generated correctly when the recommended procedure is followed.



Channel 1: PWM signal generated
Channel 2: Events

Figure 3. Timer Correct Initialization

PWM Generation Without CPU Load

In many applications, it is useful to be able to generate a PWM signal and then not have to worry about it. For example, to control LEDs illuminating a dashboard, the duty cycle and frequency do not have to be constantly changed or monitored.

The HC08 timer can generate such a signal, without loading the MCU. This mode is called unbuffered PWM generation. After the setup is done, the PWM signal is generated whatever is being executed (except Reset, Break, and if the timer is stopped), without requiring any interrupt or instruction to be executed.

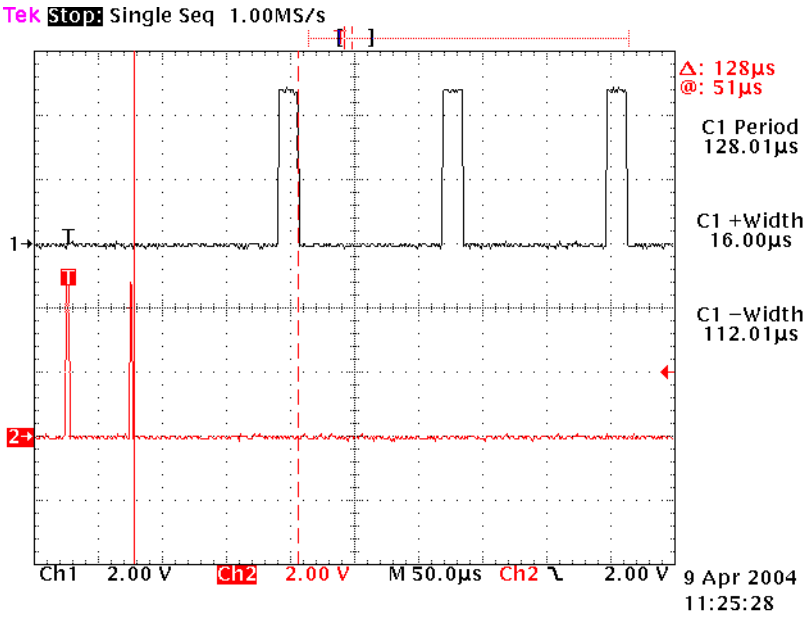
The modulo counter value represents the period of the signal. The ratio between the values in the output compare and modulo registers represents the duty cycle.

This method is simple to use and can be implemented as follows.

If the timer is configured to be reset on output compare and to toggle on overflow, the timer output state when it starts to count will be logic 0. When the output compare value is reached, the output is already at logic 0 and will only toggle to logic 1 when the first overflow occurs.

The user should be aware that the first pulse may appear to be missing.

It is possible to have this pulse earlier by inverting the signal. If the timer sets the output on output compare (bit ELSxA at logic 1 in the timer status and control register — see Table 2) instead of resetting it (keeping toggle-on-overflow bits MSxA and MSxB at logic 0), then the greater the duty cycle, the earlier the first pulse occurs.



Channel 1: PWM signal generated
Channel 2: Events

Figure 4. Set on Output Compare

The major advantage of this method is that it works separately from the CPU activity and the rest of the software.

However, it also means the timer used for the PWM generation has its modulo counter (i.e. signal period) fixed for all its other channels. Therefore, generation of multiple signals with different periods will be impossible on the same module; only different duty cycles will be possible.

Flexible PWM Rolling Generation

Principle

The solution described above is easy to set up and use, but could lack flexibility.

Using the following method will increase the possibility that the timer can generate any PWM signal on any channel. However, it will also increase the complexity and size of the code.

The principle used here is to not rely on the modulo counter to determine the period, as only one is available for all channels of the same timer.

References to the master timer modulo register, TxMODH:TxMODL, are avoided by simply adding consecutive mark (logic 1) and space (logic 0) values to the timer compare register on successive ISR function calls, as shown in the following equations. Timer roll-over is seamless when using unsigned integer addition. Using the previous compare value as a reference for generating the next compare value allows precise output timing, even though the ISR latency may vary.

$$\text{PWM Period (timer ticks)} = \frac{\text{Bus Frequency}}{\text{Timer Prescaler} \times \text{PWM Frequency}} \quad \text{Eq. 1}$$

$$\text{PWM Mark Time (timer ticks)} = \frac{\text{PWM Period} \times \text{Duty Cycle (as an integer percentage)}}{100} \quad \text{Eq. 2}$$

$$\text{PWM Space Time (timer ticks)} = \text{PWM Period} - \text{PWM Mark Time} \quad \text{Eq. 3}$$

Once the timer channel is configured, the PWM signal can be generated using the timer channel interrupt. This should be configured to call the channel interrupt service routine (ISR), loading the timer compare register with its current value added to the length desired (mark or space). This is achieved by identifying whether the last action was a negative or positive edge (the port pin toggles on output compare), and loading the compare register with the next appropriate value.

If the port pin is at logic one (or zero), a mark (or space) is to be added.

NOTE

In this case, the overflow flag and interrupt are not relevant as the counter is used as free-running with roll-over.

The sample code in the appendix at the end of this document gives an example of how to apply this principle.

Limitations

The MCU must be able to service the timer PWM interrupts in time for the next edge to be configured. This sets an upper limit on the frequency/resolution of PWM signal that can be reproduced, and will vary from system to system depending on the MCU application. It is the responsibility of the system designer to ensure that the core can update the compare registers in sufficient time using the ISR under maximum core loading conditions.

A secondary limitation is that using the timer module incurs a greater degree of core overhead, as the timer module has to be serviced at every edge transition (interrupt). This does not happen with the unbuffered PWM via modulo counter, as the toggling mechanism is independent of the core and code execution. The described method of PWM generation is likely to be more suited to slower rate PWM requirements, due to the overhead generated by having to service an interrupt for each edge of each PWM signal.

PWM Generation During Break or Software Interrupt

The HC08 data sheet states that the timer is stopped during a break. It is important to distinguish a Break from a software interrupt (SWI) instruction.

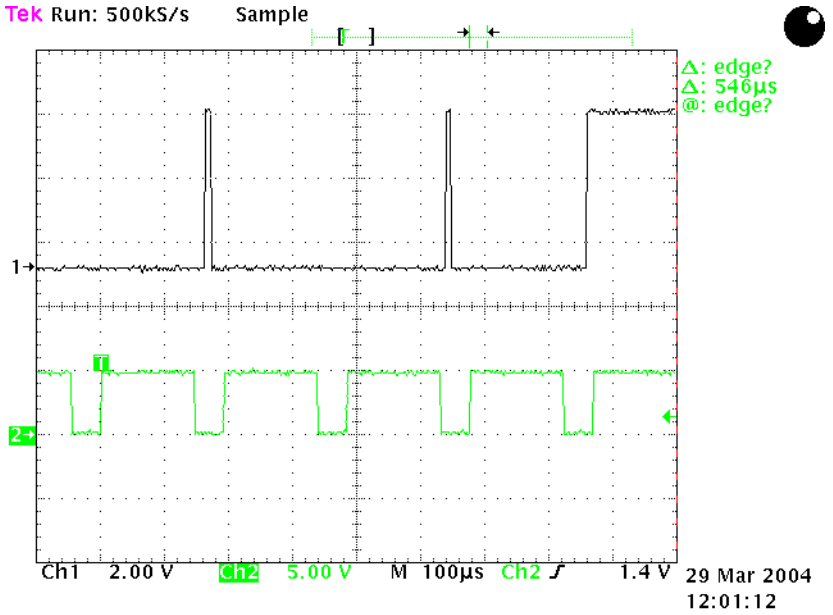
The Break Module (BRK) is primarily used in a debugging/development context. For example, breakpoints in monitor mode are generated using this module. This explains why the debugger stops program execution when an address match or BRKA bit-mask is performed.

This module could also be used in normal operation to allow for future code enhancements. For example, on a ROM device with EEPROM (like the HC08AZ60A, the GZ family does not include EEPROM), a developer can expect either to have to add a routine not available when the part is produced, or to have to manage future upgrades.

The break instruction would call the non maskable software interrupt (vector $FFFC:$FFFD), and the interrupt subroutine would reside in EEPROM.

If the program is using the timer to generate a PWM signal, the Break interrupt will hold the timer counter, to resume after its execution. Therefore, the PWM signal is affected by the interrupt, and this may present a problem if the Break interrupt execution time is significant.

Figure 5 shows how the PWM signal on Channel 1 is irregular and delayed by the activity of the interrupt (Channel 2 high when Break interrupt is being executed).



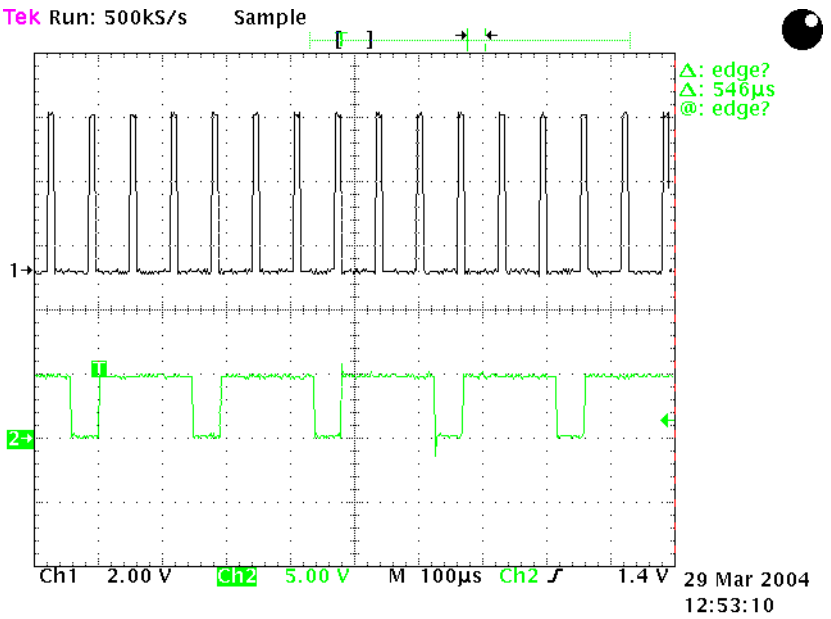
Channel 1: PWM signal generated
Channel 2: High during Break ISR execution

Figure 5. PWM Generation During a Break

Note that the user also must allow status bits to be changed (by modifying the BFCR_BCFE bit at $FE03), for them to be updated. For further details, please refer to the System Integration Module (SIM) Chapter in the device data sheet.

The Break interrupt shares its vector with the software interrupt (SWI) and the data sheet states that the Break function fetches an SWI. However, the SWI instruction does not 'freeze' any function of the MCU.

A PWM signal configured with toggle-on-overflow without using interrupt (unbuffered PWM) will not be influenced in any way by an SWI (see Figure 6).



Channel 1: PWM signal generated
Channel 2: High during SWI ISR

Figure 6. PWM Generation During and SWI

To summarize, when this interrupt has to be used in normal operation, it is advisable to use an SWI instruction, instead of a Break, to minimize the impact on the microcontroller operations (timer and other modules).

NOTE

The Break (BRK) and Software Interrupt (SWI) share the same interrupt vector. Therefore, the effect really depends on how the ISR is called and not on its content.

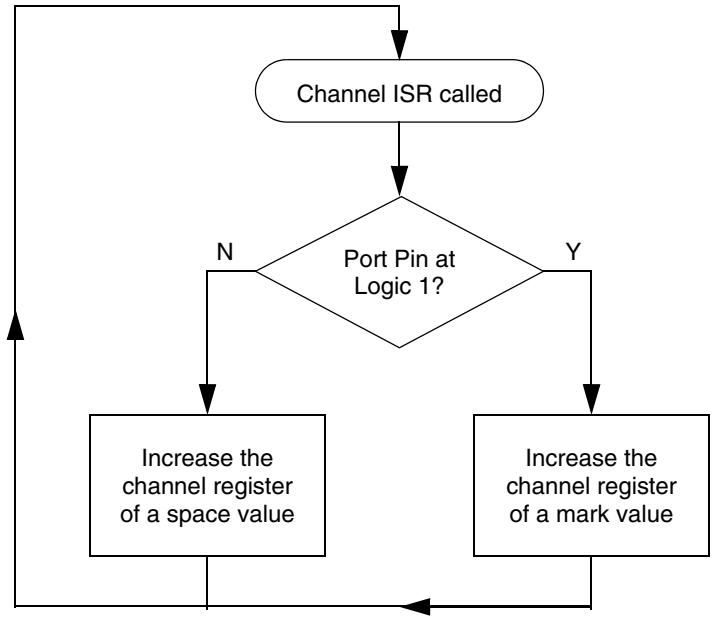
Conclusion

The HC08 timer is easy to use and can be used in a myriad of applications. However, the user must take care to set it up properly to ensure that the desired function is achieved.

Depending on the user's priority (code size, complexity, resource utilization, for example), it will be possible to find a solution to suit the application.

APPENDIX

Program Flowchart



Sample Code

```
/*****************************************************************************
*                                    COPYRIGHT (c) FREESCALE SEMICONDUCTOR 2004
*                                                          ALL RIGHTS RESERVED
* FILE NAME: AN2701_timerISR.c
*
* PURPOSE: HC908GZ60 Sample Code for the Timer Interrupt Sub Routine
*
*****************************************************************************
*****************************************************************************
**  THIS CODE IS ONLY INTENDED AS AN EXAMPLE FOR THE METROWERKS COMPILER AND **
**THE MMDS0508/EM08QA24 EVB AND HAS ONLY BEEN GIVEN A MIMIMUM LEVEL OF TEST. **
**  IT IS PROVIDED 'AS SEEN' WITH NO GUARANTEES AND NO PROMISE OF SUPPORT.   **
*****************************************************************************
*****************************************************************************
*
* DOCUMENTATION SOURCE: EKB Apps
*
* TARGET DEVICE: HC908GZ60
*
* COMPILER: Metrowerks for HC08                    VERSION: v5.2.1149
*
```

```
* DESCRIPTION: This ISR describe how to use rolling PWM generation, not to tie
*              all channels from a Timer to the same period.
*
* AUTHOR: A.RAMPON       LOCATION: EKB      LAST EDIT DATE: 15/08/04
*
* UPDATE HISTORY
* REV      AUTHOR       DATE       DESCRIPTION OF CHANGE
* ---      ------      ---------    ---------------------
* 1.0      A.Rampon     11/08/04   Initial Release
*****************************************************************************/
#ifndef __AN2701_TIMER_ISR_C
#define __AN2701_TIMER_ISR_C    /* if this H file not included, include */

#include "hc08gz60.h"

/*********************************************************************
Function Name : _PWMTimerChan0ISR
Engineer      : A.Rampon
Date          : 01/04/04
Parameters    : NONE
Returns       : NONE
Notes         : Interrupt service routine for Rolling PWM Generation on Channel 0.
*********************************************************************/
#pragma TRAP_PROC
void _PWMTimerChan0ISR(void)
{
                        /* if channel is set to generate rising edge */
                        /* On MC908AZ60A, TBCH0 is PTF4 */
                        /* If after toggled output is set, */
                        /* a mark is beginning */
    if (PTF4 == 1)
        {
                        /* set up timer compare for mark time */
                        /* relative to the last transition */
        TBCH0 += Mark[0];
        }
    else
        {
                        /* set up timer compare for space time */
                        /* relative to the last transition */
        TBCH0 += Space[0];
        }
}
```

How to Reach Us:

USA/Europe/Locations not listed:
Freescale Semiconductor Literature Distribution
P.O. Box 5405, Denver, Colorado 80217
1-800-521-6274 or 480-768-2130

Japan:
Freescale Semiconductor Japan Ltd.
SPS, Technical Information Center
3-20-1, Minami-Azabu
Minato-ku
Tokyo 106-8573, Japan
81-3-3440-3569

Asia/Pacific:
Freescale Semiconductor H.K. Ltd.
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
852-26668334

Learn More:
For more information about Freescale Semiconductor products, please visit
http://www.freescale.com

AN2701
Rev. 0, 10/2004

freescale™
semiconductor